Pattern Search

Page application

Mon, Apr 21, 2003

From time to time, there is a need for searching the front end databases for instances of particular patterns. For example, search for all analog alarm flags with a certain bit set. This note explores the idea of a page application that can do such pattern searches generically.

How should a pattern be specified? Consider a system table# to be searched. Each entry can be searched specifying an offset and a matching value and mask. In addition, one could look for matches or for mismatches. One also needs to specify a collection of nodes to be searched. An easy way to do the latter is to use a node number data file that contains the list of nodes to be searched. There are already several such data files in use, especially by the Print Memory page application PMEM, with names such as DATANLIN, DATANMRF, DATANBLM. There may be common types of searches for which the various search fields can be specified automatically, such as a search on device name wildcards.

The sequential processing of the program would be similar to that of PAGEPMEM. For each node in the data file, acquire all the relevant fields of the table specified and perform the matching algorithm. For each match, print out the node and table entry number, such as analog channel number, and the data found there. For each node, it is necessary to first find out the number of entries in the target table before requesting the relevant fields of all entries in the table. We may want to specify a range of entry numbers to be searched rather than always searching every entry.

```
      0123456789012345678901234567890123
0       PATTERN SEARCH  04/02/03 1523
1     NODES<    >           LIST<    >
2     TABLE<    >     ADDRS<         >
3     E#MIN<    >     #RECS<    >
4     #ENTS<    >     RSIZE<    >
5     EOFFS<    >
6     H—PAT<                       >
7     PMASK<                       >
8           ^   ^   ^   ^   ^   ^
9     C—PAT<                       >
10
11    *MATCH EQ
12    NODE#=
13    ENTRY=
14    EDATA=
15
```

The NODES field can be a single node number or a data file name that begins with N. The TABLE field is a decimal table# in the range 0–30. The E#MIN and #ENTS fields specify the range of entries to be searched. If #ENTS is zero, assume that all entries starting at E#MIN until the end are to be searched. The EOFFS field specifies the offset in bytes from the start of each entry to which the pattern refers. The H—PAT specifies the search pattern in hexadecimal. The PMASK field is also specified in hex; the row of carets serves as a visual guide. Alternatively, the C—PAT field specifies the pattern in ASCII, using ? to specify "don't care." The maximum number of pattern bytes in hex is 12; in ASCII, it is 24.

Click the *MATCH line to toggle through the matching logic choices of EQ, NE, GE, LT. For each match, the resulting entry is shown in the last three lines.

Initiate a pattern search by a click in rows 1–7. Whether it is a hex or character pattern search depends upon the current mode. Establish character mode by clicking in row 9 to initiate a character search. Establish hex mode by clicking in line 6 or 7 to initiate a hex pattern search. To perform a table-based search, the TABLE input field must be nonblank. If it is blank, the ADDRS, #RECS, and RSIZE fields are checked, and if valid, an array of records is searched as specified by these three fields; if the fields are invalid, no search is initiated. Whether the table-based mode is used or not, the E#MIN and #ENTS and EOFFS fields are used, just as if a table specification was given by the ADDRS, #RECS, and RSIZE fields. The mode of pattern matching may be hex or ASCII. The hex pattern mode is shown by displaying the H—PAT line prompt as H*PAT. The character pattern mode displays the C—PAT prompt as C*PAT.

The size of the pattern in hex is half the number of hex characters, of course. The size of the pattern in the ASCII mode is specified by the cursor position or the > character. The current size in that mode is shown by the position of the > character.

During execution, the program accesses the memory using information it first obtains from the node's system table directory, which yields the number of entries in the table, the size of each entry, and the base address of the table. From this information, it can make requests using one of the memory access listypes. It may require more than one request to access everything that is to be matched, depending upon the total amount of memory to be scanned. A rule of thumb might be to deal with requests or reply messages that fit within 1K bytes.

Some fields are hex, and some fields are decimal. One way to allow either to be used is to check whether the first digit is zero, which would imply hex; otherwise, it would be decimal. There are no tables, so far, that have more than 4096 entries such that this interpretation can be ambiguous. In turn, the program would need to remember which form of entry was made so that it could display it appropriately. Fields that may be entered in either form are E#MIN, #ENTS, EOFFS, #RECS, and RSIZE. The NODES field may be either a data file name, in which the first character is always N, or an individual node number in hex.

The listing generated by the program should show the node#, the entry#, and the contents of the entry, the same as is displayed at the bottom of the screen. The listing heading might be:

```
Node Ent# Address  Data EQ
```

The node# is in hex. The table# is in decimal. The entry#, address, and data are in hex. The exception is when the character mode is used, in which the data is displayed in ASCII. The latch logic is indicated at the end.

Page applications keep various parameters across invocations of the application. Those needed here are the following:

Field	*Size*	*Meaning*
fileName	4	NODES field, either name or node# in hex ascii
tableNum	2	System table number
entryMin	2	Entry# minimum
nEnts	2	#entries/records
eOffs	2	Byte offset within entry/record
arrayAddr	4	Base address of array of records
arrayRecs	2	#records in array

```
arrayRSiz    2    array record size
hexPat      12    hex pattern
hexMsk      12    hex pattern mask
charPat     24    character pattern
cPatSize     2    #characters in charPat
hPatSize     2    #bytes in hexPat pattern
listNode     2    listing node#
arrayFlag    1    array of records flag (1 = array)
match        1    match logic (0=EQ, 1=NE, 2=GE, 3=LT)
charFlag     1    character mode flag (1 = char)
hDecEMin     1    hex/decimal flags (1 = decimal)
hDecNEnt     1
hDecEOff     1
hDecNRec     1
hDecRSiz     1
```

The above variables fit well within the 120 bytes available for each page application in its PAGEM table entry. One also needs a 4-byte pointer to the static memory block that is allocated and used by the application while it is active.

As for building a request to collect the data, it is necessary to set up an array of memory address idents, each of which is 6 bytes leaving room for about 160 addresses in a 1K-byte request. If one needs more than 6 bytes from each, one would have to use less than that, in order not to exceed 1K bytes in a reply message. So, in a case of matching against analog channel names, if one needs to scan all 1024 entries, it might require about 6 cycles at 15 Hz, or about a half second. Scanning through 20 nodes would require about 10 seconds.

What about trying to scan for all recent analog descriptors? This would easily be handled if we could compare for greater than or equal, or less than, rather then merely matching or mismatching. As the bytes of data are compared, it would be better to consider all bytes as unsigned for this comparison algorithm. An exception may be made for the case of a 2-byte hex pattern. The match parameter could have 4 values that correspond to EQ, NE, GE, or LT.

Take a simple example of a search for names that look like QPS???. The input fields would be set up as follows:

```
Field             Value    Meaning
TABLE                 1    ADESC table#
E#MIN                 0    Start with entry#0
#ENTS                 0    Defaults to searching all entries
EOFFS                50    Offset to ADESC name field
C—PAT           QPS???
*MATCH EQ
```

Post-implementation

The above pattern search idea has been implemented as the local application PATS. Including some execution time diagnostics showed typical times of 4 ms when performing all the logic for up to 160 idents every 15 Hz cycle. The keyboard interrupt call that sets up a new search might take 8 ms. Page initialization takes 7 ms.